autogluon_each_location

October 9, 2023

```python
[1]: # config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*30
presets = 'best_quality'

do_drop_ds = True

use_groups = False
n_groups = 8

auto_stack = True
num_stack_levels = 1
num_bag_folds = 0
if auto_stack:
    num_stack_levels = None
    num_bag_folds = None

use_tune_data = False
use_test_data = True
tune_and_test_length = 24*30*3 # 3 months from end, this changes the␣
 ↪evaluations for only test
holdout_frac = None
use_bag_holdout = False # Enable this if there is a large gap between score_val␣
 ↪and score_test in stack models.

sample_weight = 'sample_weight' #None
weight_evaluation = True #False
sample_weight_estimated = 1 # this changes evaluations for test and tune WTF,␣
 ↪cant find a fix

run_analysis = False
```

```python
[2]: import pandas as pd
import numpy as np
```

```python
import warnings
warnings.filterwarnings("ignore")

def fix_datetime(X, name):
    # Convert 'date_forecast' to datetime format and replace original column
    ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)

    # Drop rows where the minute part of the time is not 0
    X = X[X.index.minute == 0].copy()
    return X



def convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_datetime(X_train_observed, "X_train_observed")
    X_train_estimated = fix_datetime(X_train_estimated, "X_train_estimated")
    X_test = fix_datetime(X_test, "X_test")

    # add sample weights, which are 1 for observed and 3 for estimated
    X_train_observed["sample_weight"] = 1
    X_train_estimated["sample_weight"] = sample_weight_estimated
    X_test["sample_weight"] = sample_weight_estimated

    X_train_observed["estimated_diff_hours"] = 0
    X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index - pd.
    ↪to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
    X_test["estimated_diff_hours"] = (X_test.index - pd.
    ↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

    X_train_estimated["estimated_diff_hours"] =␣
    ↪X_train_estimated["estimated_diff_hours"].astype('int64')
    # the filled once will get dropped later anyways, when we drop y nans
    X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].fillna(-50).
    ↪astype('int64')


    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)

    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
```

```python
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,
 ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =
 ↪convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train)

    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])

    # fill missng sample_weight with 3
    #X_train["sample_weight"] = X_train["sample_weight"].fillna(0)


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,
 ↪right_index=True)

    # print number of nans in sample_weight
    print(f"Number of nans in sample_weight: {X_train['sample_weight'].isna().
 ↪sum()}")
    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
```

```python
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
 ↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
Number of nans in sample_weight: 0
Number of nans in y: 0
Processing location B…
Number of nans in sample_weight: 0
Number of nans in y: 4
Processing location C…
Number of nans in sample_weight: 0
Number of nans in y: 6059
```

1 Feature enginering

```python
[3]: import numpy as np
import pandas as pd

X_train.dropna(subset=['y'], inplace=True)

if not do_drop_ds:
    # add hour datetime feature
    X_train["hour"] = X_train.index.hour
    X_test["hour"] = X_test.index.hour

#print(X_train.head())
```

```python
if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name
 ↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),
 ↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())




to_drop = ["snow_drift:idx", "snow_density:kgm3"]

X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```python
[4]: from autogluon.tabular import TabularDataset, TabularPredictor
     from autogluon.timeseries import TimeSeriesDataFrame
     import numpy as np
     train_data = TabularDataset('X_train_raw.csv')
```

```python
# set group column of train_data be increasing from 0 to 7 based on time, the
 ↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
train_data['ds'] = pd.to_datetime(train_data['ds'])
train_data = train_data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
split_time = pd.to_datetime(train_data["ds"]).max() - pd.
 ↪Timedelta(hours=tune_and_test_length)
train_set = TabularDataset(train_data[train_data["ds"] < split_time])
test_set = TabularDataset(train_data[train_data["ds"] >= split_time])
if use_groups:
    test_set = test_set.drop(columns=['group'])

if do_drop_ds:
    train_set = train_set.drop(columns=['ds'])
    test_set = test_set.drop(columns=['ds'])
    train_data = train_data.drop(columns=['ds'])


def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /
 ↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

tuning_data = None
if use_tune_data:
    train_data = train_set
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            loc_tuning_data = loc_test_set.iloc[:len(loc_test_set)//2]
            loc_test_data = loc_test_set.iloc[len(loc_test_set)//2:]
            tuning_data.append(loc_tuning_data)
            test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
```

```
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.
 ⌐shape[0], tuning_data.shape[0] + test_data.shape[0])

    else:
        tuning_data = test_set
        print("Shape of tuning", tuning_data.shape[0])

    # ensure sample weights for your tuning data sum to the number of rows in⌐
 ⌐the tuning data.
    tuning_data = normalize_sample_weights_per_location(tuning_data)


else:
    if use_test_data:
        train_data = train_set
        test_data = test_set
        print("Shape of test", test_data.shape[0])

# ensure sample weights for your training (or tuning) data sum to the number of⌐
 ⌐rows in the training (or tuning) data.
train_data = normalize_sample_weights_per_location(train_data)
if use_test_data:
    test_data = normalize_sample_weights_per_location(test_data)
```

Shape of test 5791

```
[5]: if run_analysis:
        import autogluon.eda.auto as auto
        auto.dataset_overview(train_data=train_data, test_data=test_data,⌐
 ⌐label="y", sample=None)
```

```
[6]: if run_analysis:
        auto.target_analysis(train_data=train_data, label="y")
```

2 Starting

```
[7]: import os


    # Get the last submission number
    last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for⌐
 ⌐filename in os.listdir('submissions') if "submission" in filename]))
    print("Last submission number:", last_submission_number)
    print("Now creating submission number:", last_submission_number + 1)

    # Create the new filename
    new_filename = f'submission_{last_submission_number + 1}'
```

```python
hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 81
Now creating submission number: 82
New filename: submission_82
```

```
[8]: predictors = [None, None, None]
```

```python
[9]: def fit_predictor_for_location(loc):
         print(f"Training model for location {loc}...")
         # sum of sample weights for this location, and number of rows, for both
      ↪train and tune data and test data
         print("Train data sample weight sum:", train_data[train_data["location"] ==
      ↪loc]["sample_weight"].sum())
         print("Train data number of rows:", train_data[train_data["location"] ==
      ↪loc].shape[0])
         if use_tune_data:
             print("Tune data sample weight sum:",
      ↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
             print("Tune data number of rows:", tuning_data[tuning_data["location"]
      ↪== loc].shape[0])
         if use_test_data:
             print("Test data sample weight sum:", test_data[test_data["location"]
      ↪== loc]["sample_weight"].sum())
             print("Test data number of rows:", test_data[test_data["location"] ==
      ↪loc].shape[0])
         predictor = TabularPredictor(
             label=label,
             eval_metric=metric,
             path=f"AutogluonModels/{new_filename}_{loc}",
             sample_weight=sample_weight,
             weight_evaluation=weight_evaluation,
             groups="group" if use_groups else None,
         ).fit(
             train_data=train_data[train_data["location"] == loc],
             time_limit=time_limit,
             #presets=presets,
             num_stack_levels=num_stack_levels,
             num_bag_folds=num_bag_folds if not use_groups else 2,# just put
      ↪somethin, will be overwritten anyways
             tuning_data=tuning_data[tuning_data["location"] == loc] if
      ↪use_tune_data else None,
```

```
        use_bag_holdout=use_bag_holdout,
        holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

Values in column 'sample_weight' used as sample weights instead of predictive
features. Evaluation will report weighted metrics, so ensure same column exists
in test data.

Training model for location A…
Train data sample weight sum: 31900
Train data number of rows: 31900
Test data sample weight sum: 2161
Test data number of rows: 2161

Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_82_A/"
AutoGluon Version: 0.8.2
Python Version: 3.10.12
Operating System: Linux
Platform Machine: x86_64
Platform Version: #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail: 306.74 GB / 315.93 GB (97.1%)
Train Data Rows: 31900
Train Data Columns: 46
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 633.132, 1165.64686)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory: 132278.92 MB

```
        Train Data (Original)  Memory Usage: 13.08 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 3 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 42 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['estimated_diff_hours']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])       :  1 | ['estimated_diff_hours']
                ('int', ['bool']) :  3 | ['is_day:idx', 'is_in_shadow:idx',
'wind_speed_w_1000hPa:ms']
        0.2s = Fit runtime
        43 features in original data used to generate 43 features in processed
data.
        Train Data (Processed) Memory Usage: 10.3 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.19s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
Automatically generating train/validation split with
holdout_frac=0.07836990595611286, Train Rows: 29400, Val Rows: 2500
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
```

```
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif … Training model for up to 1799.81s of the
1799.8s of remaining time.
        -285.3795       = Validation score   (-mean_absolute_error)
        0.04s    = Training   runtime
        0.06s    = Validation runtime
Fitting model: KNeighborsDist … Training model for up to 1799.7s of the
1799.69s of remaining time.
        -288.0059       = Validation score   (-mean_absolute_error)
        0.04s    = Training   runtime
        0.04s    = Validation runtime
Fitting model: LightGBMXT … Training model for up to 1799.61s of the 1799.61s
of remaining time.

[1000]  valid_set's l1: 178.37
[2000]  valid_set's l1: 174.975
[3000]  valid_set's l1: 173.514
[4000]  valid_set's l1: 172.456
[5000]  valid_set's l1: 172.017
[6000]  valid_set's l1: 171.584
[7000]  valid_set's l1: 171.168
[8000]  valid_set's l1: 170.818
[9000]  valid_set's l1: 170.597
[10000] valid_set's l1: 170.345

        -170.3454       = Validation score   (-mean_absolute_error)
        13.41s   = Training   runtime
        0.16s    = Validation runtime
Fitting model: LightGBM … Training model for up to 1785.75s of the 1785.74s of
remaining time.
```

```
[1000]  valid_set's l1: 182.44
[2000]  valid_set's l1: 180.615
[3000]  valid_set's l1: 180.212

        -180.107        = Validation score   (-mean_absolute_error)
        4.79s    = Training    runtime
        0.05s    = Validation runtime
Fitting model: RandomForestMSE … Training model for up to 1780.82s of the
1780.81s of remaining time.
        -187.2246       = Validation score   (-mean_absolute_error)
        7.57s    = Training    runtime
        0.09s    = Validation runtime
Fitting model: CatBoost … Training model for up to 1772.68s of the 1772.67s of
remaining time.
        -181.3073       = Validation score   (-mean_absolute_error)
        110.97s  = Training    runtime
        0.01s    = Validation runtime
Fitting model: ExtraTreesMSE … Training model for up to 1661.66s of the
1661.66s of remaining time.
        -186.6021       = Validation score   (-mean_absolute_error)
        1.66s    = Training    runtime
        0.08s    = Validation runtime
Fitting model: NeuralNetFastAI … Training model for up to 1659.43s of the
1659.43s of remaining time.
        -192.4111       = Validation score   (-mean_absolute_error)
        26.44s   = Training    runtime
        0.04s    = Validation runtime
Fitting model: XGBoost … Training model for up to 1632.92s of the 1632.92s of
remaining time.
        -186.0713       = Validation score   (-mean_absolute_error)
        2.32s    = Training    runtime
        0.01s    = Validation runtime
Fitting model: NeuralNetTorch … Training model for up to 1630.56s of the
1630.56s of remaining time.
        -176.1157       = Validation score   (-mean_absolute_error)
        51.73s   = Training    runtime
        0.04s    = Validation runtime
Fitting model: LightGBMLarge … Training model for up to 1578.79s of the
1578.78s of remaining time.

[1000]  valid_set's l1: 172.273
[2000]  valid_set's l1: 170.86
[3000]  valid_set's l1: 170.486
[4000]  valid_set's l1: 170.39
[5000]  valid_set's l1: 170.319
[6000]  valid_set's l1: 170.298
[7000]  valid_set's l1: 170.29
[8000]  valid_set's l1: 170.284
[9000]  valid_set's l1: 170.282
```

```
[10000] valid_set's l1: 170.28

        -170.2803        = Validation score   (-mean_absolute_error)
        43.51s   = Training   runtime
        0.26s    = Validation runtime
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
1533.87s of remaining time.
        -163.223         = Validation score   (-mean_absolute_error)
        0.46s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 266.63s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_82_A/")
WARNING: eval_metric='pearsonr' does not support sample weights so they will be
ignored in reported metric.
Evaluation: mean_absolute_error on test data: -187.8065158485432
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -187.8065158485432,
    "root_mean_squared_error": -414.32909739992886,
    "mean_squared_error": -171668.60095223974,
    "r2": 0.8754434810346822,
    "pearsonr": 0.9358470155799331,
    "median_absolute_error": -12.917183456420899
}

Evaluation on test data:
-187.8065158485432
```

```
[10]: loc = "B"
      predictors[1] = fit_predictor_for_location(loc)
```

```
Values in column 'sample_weight' used as sample weights instead of predictive
features. Evaluation will report weighted metrics, so ensure same column exists
in test data.
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_82_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   305.78 GB / 315.93 GB (96.8%)
Train Data Rows:    30768
Train Data Columns: 46
Label Column: y
```

```
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 97.74541, 195.0957)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    130496.11 MB
        Train Data (Original)  Memory Usage: 12.62 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 3 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…

Training model for location B…
Train data sample weight sum: 30768
Train data number of rows: 30768
Test data sample weight sum: 2051
Test data number of rows: 2051

        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 42 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['estimated_diff_hours']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])       :  1 | ['estimated_diff_hours']
                ('int', ['bool']) :  3 | ['is_day:idx', 'is_in_shadow:idx',
```

```
'wind_speed_w_1000hPa:ms']
        0.1s = Fit runtime
        43 features in original data used to generate 43 features in processed
data.
        Train Data (Processed) Memory Usage: 9.94 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.18s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
Automatically generating train/validation split with
holdout_frac=0.0812532501300052, Train Rows: 28268, Val Rows: 2500
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif … Training model for up to 1799.82s of the
1799.82s of remaining time.
        -57.0973            = Validation score    (-mean_absolute_error)
        0.03s    = Training    runtime
        0.04s    = Validation runtime
Fitting model: KNeighborsDist … Training model for up to 1799.74s of the
1799.73s of remaining time.
        -56.8969            = Validation score    (-mean_absolute_error)
        0.03s    = Training    runtime
        0.04s    = Validation runtime
Fitting model: LightGBMXT … Training model for up to 1799.65s of the 1799.65s
of remaining time.
```

```
[1000]   valid_set's l1: 35.5751
[2000]   valid_set's l1: 33.3902
[3000]   valid_set's l1: 32.2742
[4000]   valid_set's l1: 31.5407
[5000]   valid_set's l1: 31.0096
[6000]   valid_set's l1: 30.6243
[7000]   valid_set's l1: 30.3162
[8000]   valid_set's l1: 30.0585
[9000]   valid_set's l1: 29.8764
[10000]  valid_set's l1: 29.726
```

        -29.7259   = Validation score   (-mean_absolute_error)
        12.98s   = Training   runtime
        0.17s   = Validation runtime
Fitting model: LightGBM … Training model for up to 1786.25s of the 1786.25s of
remaining time.

```
[1000]   valid_set's l1: 33.0342
[2000]   valid_set's l1: 31.7436
[3000]   valid_set's l1: 31.1409
[4000]   valid_set's l1: 30.8249
[5000]   valid_set's l1: 30.6002
[6000]   valid_set's l1: 30.4238
[7000]   valid_set's l1: 30.3416
[8000]   valid_set's l1: 30.2763
[9000]   valid_set's l1: 30.2196
[10000]  valid_set's l1: 30.185
```

        -30.185   = Validation score   (-mean_absolute_error)
        13.4s   = Training   runtime
        0.16s   = Validation runtime
Fitting model: RandomForestMSE … Training model for up to 1772.45s of the
1772.44s of remaining time.
        -35.3114   = Validation score   (-mean_absolute_error)
        8.52s   = Training   runtime
        0.09s   = Validation runtime
Fitting model: CatBoost … Training model for up to 1763.48s of the 1763.48s of
remaining time.
        -32.7391   = Validation score   (-mean_absolute_error)
        112.41s = Training   runtime
        0.01s   = Validation runtime
Fitting model: ExtraTreesMSE … Training model for up to 1651.03s of the
1651.02s of remaining time.
        -36.4349   = Validation score   (-mean_absolute_error)
        1.73s   = Training   runtime
        0.1s   = Validation runtime
Fitting model: NeuralNetFastAI … Training model for up to 1648.8s of the
1648.79s of remaining time.
        -40.4352   = Validation score   (-mean_absolute_error)

```
        26.26s   = Training   runtime
        0.03s    = Validation runtime
Fitting model: XGBoost … Training model for up to 1622.47s of the 1622.46s of
remaining time.
        -33.3765          = Validation score   (-mean_absolute_error)
        24.23s   = Training   runtime
        0.21s    = Validation runtime
Fitting model: NeuralNetTorch … Training model for up to 1597.88s of the
1597.87s of remaining time.
        -34.0567          = Validation score   (-mean_absolute_error)
        98.56s   = Training   runtime
        0.04s    = Validation runtime
Fitting model: LightGBMLarge … Training model for up to 1499.28s of the
1499.27s of remaining time.

[1000]  valid_set's l1: 30.2342
[2000]  valid_set's l1: 29.3138
[3000]  valid_set's l1: 29.0572
[4000]  valid_set's l1: 28.983
[5000]  valid_set's l1: 28.955
[6000]  valid_set's l1: 28.9422
[7000]  valid_set's l1: 28.9365
[8000]  valid_set's l1: 28.9338
[9000]  valid_set's l1: 28.9325
[10000] valid_set's l1: 28.9318

        -28.9318          = Validation score   (-mean_absolute_error)
        41.89s   = Training   runtime
        0.28s    = Validation runtime
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
1455.96s of remaining time.
        -28.419  = Validation score   (-mean_absolute_error)
        0.45s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 344.52s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_82_B/")
WARNING: eval_metric='pearsonr' does not support sample weights so they will be
ignored in reported metric.
Evaluation: mean_absolute_error on test data: -37.12180427631004
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -37.12180427631004,
    "root_mean_squared_error": -81.58609851989235,
    "mean_squared_error": -6656.291471697579,
    "r2": 0.7859139269118017,
```

```
        "pearsonr": 0.9104600209448434,
        "median_absolute_error": -8.015130996704102
    }

    Evaluation on test data:
    -37.12180427631004
```

[11]:
```
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
```

```
Values in column 'sample_weight' used as sample weights instead of predictive
features. Evaluation will report weighted metrics, so ensure same column exists
in test data.
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_82_C/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   304.87 GB / 315.93 GB (96.5%)
Train Data Rows:    24492
Train Data Columns: 46
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
        Label info (max, min, mean, stddev): (999.6, 0.0, 78.11911, 167.50151)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    130258.45 MB
        Train Data (Original)  Memory Usage: 10.04 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
```

Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 42 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   : 1 | ['estimated_diff_hours']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])   : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   : 1 | ['estimated_diff_hours']
                ('int', ['bool']) : 2 | ['is_day:idx', 'is_in_shadow:idx']
        0.1s = Fit runtime
        43 features in original data used to generate 43 features in processed
data.

Training model for location C…
Train data sample weight sum: 24492
Train data number of rows: 24492
Test data sample weight sum: 1579
Test data number of rows: 1579

        Train Data (Processed) Memory Usage: 8.08 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.16s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
Automatically generating train/validation split with holdout_frac=0.1, Train
Rows: 22042, Val Rows: 2450
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],

```
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif … Training model for up to 1799.84s of the
1799.84s of remaining time.
        -33.2822         = Validation score    (-mean_absolute_error)
        0.03s    = Training    runtime
        0.03s    = Validation runtime
Fitting model: KNeighborsDist … Training model for up to 1799.77s of the
1799.77s of remaining time.
        -33.3446         = Validation score    (-mean_absolute_error)
        0.03s    = Training    runtime
        0.03s    = Validation runtime
Fitting model: LightGBMXT … Training model for up to 1799.7s of the 1799.69s
of remaining time.

[1000]  valid_set's l1: 18.9213
[2000]  valid_set's l1: 18.3545
[3000]  valid_set's l1: 18.1711
[4000]  valid_set's l1: 18.08
[5000]  valid_set's l1: 18.0196
[6000]  valid_set's l1: 17.9721
[7000]  valid_set's l1: 17.9335
[8000]  valid_set's l1: 17.9153
[9000]  valid_set's l1: 17.9061
[10000] valid_set's l1: 17.8961

        -17.891  = Validation score    (-mean_absolute_error)
        12.13s   = Training    runtime
        0.16s    = Validation runtime
Fitting model: LightGBM … Training model for up to 1787.16s of the 1787.15s of
remaining time.

[1000]  valid_set's l1: 19.115
[2000]  valid_set's l1: 18.8635
[3000]  valid_set's l1: 18.8186
[4000]  valid_set's l1: 18.7676
[5000]  valid_set's l1: 18.7493
[6000]  valid_set's l1: 18.7353
[7000]  valid_set's l1: 18.7294
[8000]  valid_set's l1: 18.7245
[9000]  valid_set's l1: 18.7201
[10000] valid_set's l1: 18.7209
```

```
        -18.7199          = Validation score    (-mean_absolute_error)
        12.52s    = Training    runtime
        0.15s     = Validation runtime
Fitting model: RandomForestMSE … Training model for up to 1774.27s of the
1774.26s of remaining time.
        -20.2022          = Validation score    (-mean_absolute_error)
        4.58s    = Training    runtime
        0.09s    = Validation runtime
Fitting model: CatBoost … Training model for up to 1769.44s of the 1769.43s of
remaining time.
        -18.5962          = Validation score    (-mean_absolute_error)
        112.1s    = Training    runtime
        0.01s     = Validation runtime
Fitting model: ExtraTreesMSE … Training model for up to 1657.29s of the
1657.28s of remaining time.
        -20.1925          = Validation score    (-mean_absolute_error)
        1.01s    = Training    runtime
        0.08s    = Validation runtime
Fitting model: NeuralNetFastAI … Training model for up to 1656.02s of the
1656.02s of remaining time.
        -20.3136          = Validation score    (-mean_absolute_error)
        20.44s    = Training    runtime
        0.03s     = Validation runtime
Fitting model: XGBoost … Training model for up to 1635.51s of the 1635.51s of
remaining time.
        -18.7778          = Validation score    (-mean_absolute_error)
        24.29s    = Training    runtime
        0.21s     = Validation runtime
Fitting model: NeuralNetTorch … Training model for up to 1610.86s of the
1610.86s of remaining time.
        -18.985  = Validation score    (-mean_absolute_error)
        77.38s    = Training    runtime
        0.03s     = Validation runtime
Fitting model: LightGBMLarge … Training model for up to 1533.43s of the
1533.43s of remaining time.

[1000]  valid_set's l1: 18.3614
[2000]  valid_set's l1: 18.2652
[3000]  valid_set's l1: 18.2436
[4000]  valid_set's l1: 18.238
[5000]  valid_set's l1: 18.2362
[6000]  valid_set's l1: 18.2354
[7000]  valid_set's l1: 18.2351
[8000]  valid_set's l1: 18.235
[9000]  valid_set's l1: 18.235
[10000] valid_set's l1: 18.235

        -18.235  = Validation score    (-mean_absolute_error)
        44.45s    = Training    runtime
```

```
        0.32s     = Validation runtime
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
1487.5s of remaining time.
        -17.067  = Validation score    (-mean_absolute_error)
        0.45s     = Training    runtime
        0.0s      = Validation runtime
AutoGluon training complete, total runtime = 312.98s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_82_C/")
WARNING: eval_metric='pearsonr' does not support sample weights so they will be
ignored in reported metric.
Evaluation: mean_absolute_error on test data: -30.572125653402658
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -30.572125653402658,
    "root_mean_squared_error": -63.303145108710055,
    "mean_squared_error": -4007.2881806544015,
    "r2": 0.7898971621150155,
    "pearsonr": 0.8943104148622111,
    "median_absolute_error": -3.0131614208221436
}

Evaluation on test data:
-30.572125653402658
```

3 Submit

```python
[12]: import pandas as pd
      import matplotlib.pyplot as plt

      train_data_with_dates = TabularDataset('X_train_raw.csv')
      train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

      test_data = TabularDataset('X_test_raw.csv')
      test_data["ds"] = pd.to_datetime(test_data["ds"])
      #test_data
```

```
Loaded data from: X_train_raw.csv | Columns = 48 / 48 | Rows = 92951 -> 92951
Loaded data from: X_test_raw.csv | Columns = 47 / 47 | Rows = 2160 -> 2160
```

```python
[13]: test_ids = TabularDataset('test.csv')
      test_ids["time"] = pd.to_datetime(test_ids["time"])
      # merge test_data with test_ids
      test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",␣
        ↪"location"], left_on=["ds", "location"])
```

```
#test_data_merged
```

Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160

```python
[14]: # predict, grouped by location
      predictions = []
      location_map = {
          "A": 0,
          "B": 1,
          "C": 2
      }
      for loc, group in test_data.groupby('location'):
          i = location_map[loc]
          subset = test_data_merged[test_data_merged["location"] == loc].
       ↪reset_index(drop=True)
          #print(subset)
          pred = predictors[i].predict(subset)
          subset["prediction"] = pred
          predictions.append(subset)

          # get past predictions
          past_pred = predictors[i].
       ↪predict(train_data_with_dates[train_data_with_dates["location"] == loc])
          train_data_with_dates.loc[train_data_with_dates["location"] == loc,␣
       ↪"prediction"] = past_pred
```

```python
[15]: # plot predictions for location A, in addition to train data for A
      for loc, idx in location_map.items():
          fig, ax = plt.subplots(figsize=(20, 10))
          # plot train data
          train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
       ↪y='y', ax=ax, label="train data")

          # plot predictions
          predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

          # plot past predictions
          train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
       ↪y='prediction', ax=ax, label="past predictions")

          # title
          ax.set_title(f"Predictions for location {loc}")
```

Predictions for location A

Predictions for location B

Predictions for location C
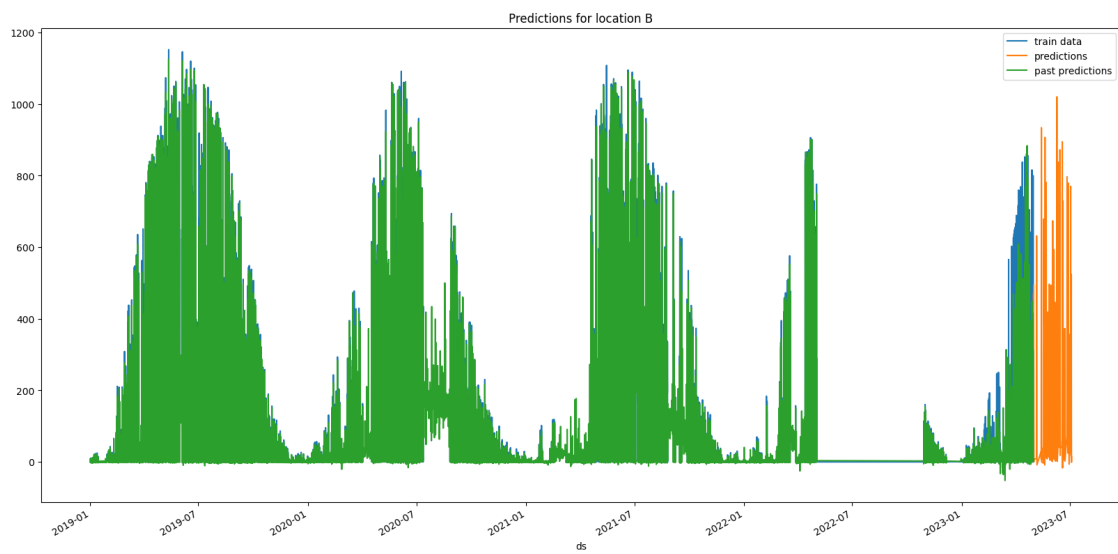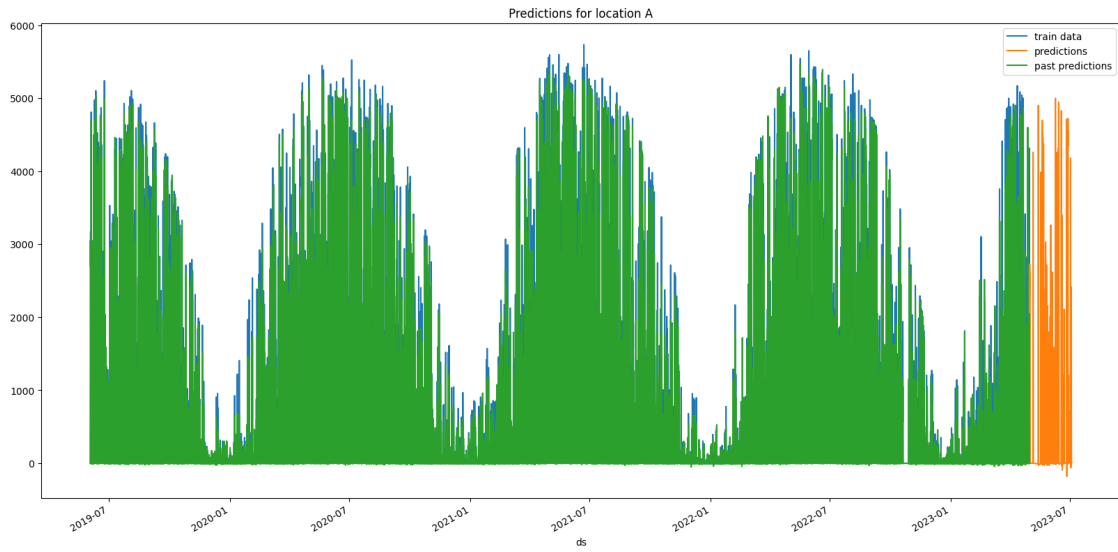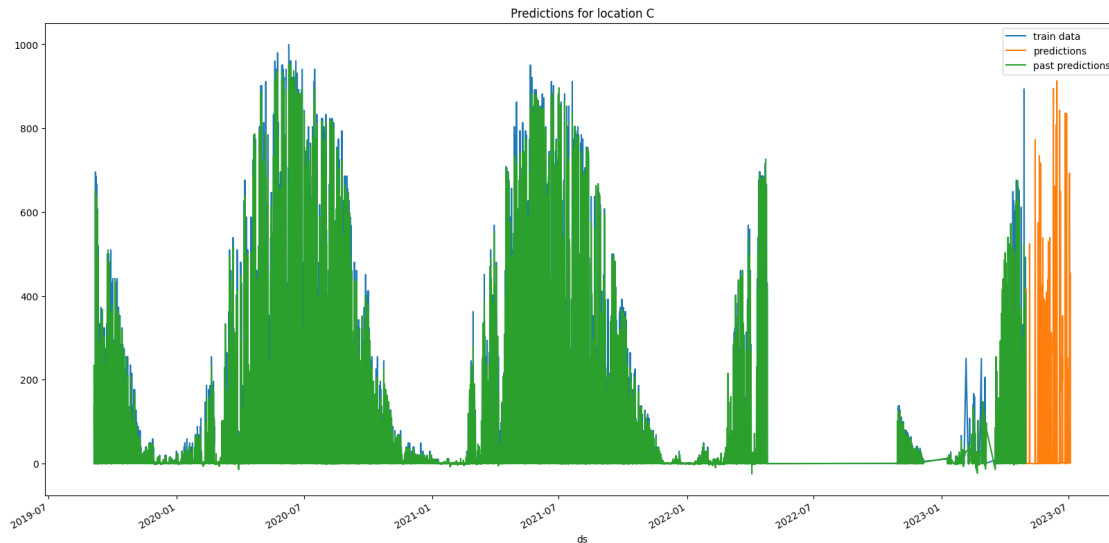
```python
[16]:  # concatenate predictions
       submissions_df = pd.concat(predictions)
       submissions_df = submissions_df[["id", "prediction"]]
       submissions_df
```

```
[16]:        id   prediction
       0       0    -2.451493
       1       1    -2.136074
       2       2    -0.487430
       3       3    44.617561
       4       4   357.281982
       ..    ...          ...
       715   2155    57.938728
       716   2156    36.972794
       717   2157    10.466211
       718   2158     1.646137
       719   2159     1.249219

       [2160 rows x 2 columns]
```

```python
[17]:  # Save the submission DataFrame to submissions folder, create new name based on␣
       ↪last submission, format is submission_<last_submission_number + 1>.csv

       # Save the submission
       print(f"Saving submission to submissions/{new_filename}.csv")
       submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),␣
       ↪index=False)
       print("jall1a")
```

```
Saving submission to submissions/submission_82.csv
jall1a
```

[18]:
```python
# save this running notebook
from IPython.display import display, Javascript
import time

# hei123

display(Javascript("IPython.notebook.save_checkpoint();"))

time.sleep(3)
```

```
<IPython.core.display.Javascript object>
```

[19]:
```python
# save this notebook to submissions folder
import subprocess
import os
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
  ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
  ↪ipynb"])
```

```
[NbConvertApp] Converting notebook autogluon_each_location.ipynb to pdf
/opt/conda/lib/python3.10/site-packages/nbconvert/utils/pandoc.py:51:
RuntimeWarning: You are using an unsupported version of pandoc (2.9.2.1).
Your version must be at least (2.14.2) but less than (4.0.0).
Refer to https://pandoc.org/installing.html.
Continuing with doubts…
  check_pandoc_version()
[NbConvertApp] Writing 108539 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 88398 bytes to notebook_pdfs/submission_82.pdf
```

[19]:
```
CompletedProcess(args=['jupyter', 'nbconvert', '--to', 'pdf', '--output',
  'notebook_pdfs/submission_82.pdf', 'autogluon_each_location.ipynb'],
  returncode=0)
```

[20]:
```python
# feature importance
location="A"
split_time = pd.Timestamp("2022-10-28 22:00:00")
estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
estimated = estimated[estimated["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=estimated,
  ↪time_limit=60*10)
```

These features in provided data are not utilized by the predictor and will be ignored: ['ds', 'elevation:m', 'sample_weight', 'location', 'prediction']
Computing feature importance via permutation shuffling for 43 features using 4394 rows with 10 shuffle sets… Time limit: 600s…
        527.39s = Expected runtime (52.74s per shuffle set)
        297.97s = Actual runtime (Completed 10 of 10 shuffle sets)

[20]:

	importance	stddev	p_value	n \
direct_rad:W	147.111766	2.176199	2.728679e-18	10
clear_sky_rad:W	88.789385	1.940390	9.137558e-17	10
diffuse_rad:W	78.019292	2.133635	6.869250e-16	10
sun_azimuth:d	56.670397	3.018095	2.746007e-13	10
sun_elevation:d	33.984099	0.972222	1.030239e-15	10
direct_rad_1h:J	28.152551	0.600915	7.392124e-17	10
clear_sky_energy_1h:J	27.416397	1.190033	4.372890e-14	10
diffuse_rad_1h:J	15.161300	0.741926	1.284197e-13	10
total_cloud_cover:p	14.086562	0.636443	6.268596e-14	10
effective_cloud_cover:p	12.502470	0.835651	2.108495e-12	10
wind_speed_u_10m:ms	9.767501	1.054446	1.536273e-10	10
cloud_base_agl:m	7.589929	0.452026	7.490431e-13	10
is_day:idx	6.359774	0.328628	2.093811e-13	10
visibility:m	5.828035	0.524073	3.004598e-11	10
snow_water:kgm2	5.682946	0.834743	2.368539e-09	10
relative_humidity_1000hPa:p	5.646109	0.588591	1.124939e-10	10
ceiling_height_agl:m	5.453996	0.415181	6.765571e-12	10
fresh_snow_24h:cm	5.347722	0.684959	7.034902e-10	10
msl_pressure:hPa	5.049854	0.569148	2.255062e-10	10
wind_speed_10m:ms	4.526295	0.654019	2.048277e-09	10
is_in_shadow:idx	4.427495	0.273039	1.024165e-12	10
pressure_50m:hPa	4.083450	0.569790	1.503712e-09	10
wind_speed_v_10m:ms	4.031230	0.700483	1.041920e-08	10
sfc_pressure:hPa	3.598798	0.589320	6.183099e-09	10
pressure_100m:hPa	3.595549	0.573325	4.890576e-09	10
t_1000hPa:K	2.035832	1.175175	1.955336e-04	10
estimated_diff_hours	1.725568	0.172421	7.706771e-11	10
fresh_snow_6h:cm	1.596119	0.229132	1.933599e-09	10
fresh_snow_12h:cm	1.518177	0.306898	3.914704e-08	10
air_density_2m:kgm3	1.361581	0.550849	1.331568e-05	10
super_cooled_liquid_water:kgm2	1.205138	0.336866	6.350857e-07	10
snow_depth:cm	1.185776	0.372307	1.685825e-06	10
precip_5min:mm	0.986953	0.423080	2.102601e-05	10
fresh_snow_3h:cm	0.847326	0.204491	1.814081e-07	10
dew_point_2m:K	0.682033	0.427291	3.463186e-04	10
precip_type_5min:idx	0.587061	0.292100	6.600589e-05	10
dew_or_rime:idx	0.516153	0.187415	5.580908e-06	10
fresh_snow_1h:cm	0.400973	0.194604	5.471623e-05	10
rain_water:kgm2	0.162171	0.113489	7.247112e-04	10

```
prob_rime:p                          0.045152  0.143556  1.729571e-01  10
wind_speed_w_1000hPa:ms              0.000000  0.000000  5.000000e-01  10
snow_melt_10min:mm                  -0.104706  0.110621  9.924385e-01  10
absolute_humidity_2m:gm3            -0.262476  0.207911  9.984264e-01  10


                                      p99_high    p99_low
direct_rad:W                        149.348220  144.875311
clear_sky_rad:W                      90.783502   86.795269
diffuse_rad:W                        80.212004   75.826580
sun_azimuth:d                        59.772058   53.568736
sun_elevation:d                      34.983240   32.984958
direct_rad_1h:J                      28.770104   27.534998
clear_sky_energy_1h:J                28.639380   26.193414
diffuse_rad_1h:J                     15.923768   14.398832
total_cloud_cover:p                  14.740627   13.432497
effective_cloud_cover:p              13.361259   11.643682
wind_speed_u_10m:ms                  10.851142    8.683859
cloud_base_agl:m                      8.054470    7.125387
is_day:idx                            6.697500    6.022047
visibility:m                          6.366619    5.289451
snow_water:kgm2                       6.540801    4.825090
relative_humidity_1000hPa:p           6.250998    5.041221
ceiling_height_agl:m                  5.880672    5.027319
fresh_snow_24h:cm                     6.051646    4.643798
msl_pressure:hPa                      5.634761    4.464948
wind_speed_10m:ms                     5.198422    3.854168
is_in_shadow:idx                      4.708094    4.146896
pressure_50m:hPa                      4.669017    3.497883
wind_speed_v_10m:ms                   4.751108    3.311352
sfc_pressure:hPa                      4.204435    2.993161
pressure_100m:hPa                     4.184749    3.006349
t_1000hPa:K                           3.243546    0.828118
estimated_diff_hours                  1.902763    1.548373
fresh_snow_6h:cm                      1.831595    1.360643
fresh_snow_12h:cm                     1.833572    1.202782
air_density_2m:kgm3                   1.927683    0.795480
super_cooled_liquid_water:kgm2        1.551332    0.858945
snow_depth:cm                         1.568392    0.803160
precip_5min:mm                        1.421747    0.552159
fresh_snow_3h:cm                      1.057479    0.637173
dew_point_2m:K                        1.121155    0.242911
precip_type_5min:idx                  0.887249    0.286873
dew_or_rime:idx                       0.708757    0.323548
fresh_snow_1h:cm                      0.600965    0.200980
rain_water:kgm2                       0.278803    0.045539
prob_rime:p                           0.192682   -0.102379
wind_speed_w_1000hPa:ms               0.000000    0.000000
```

```
snow_melt_10min:mm                0.008979    -0.218390
absolute_humidity_2m:gm3         -0.048808    -0.476143
```

```python
# feature importance
observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
observed = observed[observed["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=observed,
 ↪time_limit=60*10)
```

These features in provided data are not utilized by the predictor and will be ignored: ['ds', 'elevation:m', 'sample_weight', 'location', 'prediction']
Computing feature importance via permutation shuffling for 43 features using 5000 rows with 10 shuffle sets… Time limit: 600s…
        594.31s = Expected runtime (59.43s per shuffle set)

```python
display(Javascript("IPython.notebook.save_checkpoint();"))
time.sleep(3)

subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),
 ↪"autogluon_each_location.ipynb"])
```

```python
# import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,
 ↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
 ↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',
 ↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is
 ↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"
```

```
# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',␣
 ↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
 ↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```